Jari Eikkula

Tiedon välittäminen Raspberry Pi -koneelta palvelimen kautta Linux-, Windows- ja Android-käyttäjille.

Opinnäytetyö
Kevät 2015
SeAMK Tekniikka
Tietotekniikan Tutkinto-ohjelma

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Tekniikanyksikkö

Tutkinto-ohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Jari Eikkula

Työn nimi: Tiedon välittäminen Raspberry Pi -koneelta palvelimen kautta Linux-, Windows- ja Android-käyttäjille.

Ohjaaja: Mäkelä Petteri

Vuosi: 2015 Sivumäärä: 45 Liitteiden lukumäärä: 0

Työssä on toteutettu tiedonvälitysohjelmiston prototyyppi. Ohjelmisto perustuu asiakas-palvelin-malliin, ja on toteutettu käyttäen Publish-Subscribe-ohjelmisto-mallia. Ohjelmisto koostuu tietoa lähettävistä ja vastaanottavista asiakasohjelmista sekä palvelinohjelmasta.

Työssä on testattu myös avoimen lähdekoodin Mono .Net framework -ohjelmistoa, jolla Window-ympäristössä käännetty C#-kielinen binääriohjelma voidaan ajaa sellaisenaan Linux-ympäristössä. Asiakasohjelmista on toteutettu C#-kielellä sekä Windows Forms- että komentoriviversiot. Näitä asiakasohjelmia sitten testattiin sekä Windows-koneessa sellaisenaan että Linux-koneessa Mono-ohjelmistolla. Lisäksi asiakasohjelmista toteutettiin myös Python-kieliset asiakasohjelmat, joita voidaan myös käyttää sellaisenaan sekä Windows- että Linux-ympäristössä. Subscriber-asiakkas toteutettiin myös Android-puhelimille ja -tableteille Java-kielellä.

Avainsanat: verkko-ohjelmointi, Linux, Windows, Android, C#, Python, Java, avoin lähdekoodi, Publish-Subscribe-ohjelmistomalli, Mono

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Software

Author: Jari Eikkula

Title of thesis: Delivering information from Raspberry Pi computer through a server to Linux, Windows and Android clients

Supervisor: Petteri Mäkelä

Year: 2015 Number of pages: 45 Number of appendices: 0

In this thesis there was implemented a prototype of an information delivery system. The implementation is a client-server system based on the Publish-Subscribe pattern. The software consists of a client sending the information, another client receiving the information and a server delivering the information between these two client types.

One target of the thesis was to test an open source framework called Mono. It can be used to run the C#-binary code that was developed on Windows-system also on the Linux system. In this thesis there were implemented two versions of these clients: a Windows Forms based version for Windows environments and a command line version for text based environments. These clients were tested as such on the Windows-system using binary code. The same clients where tested also on the Linux-system using the Mono framework. In addition, the clients were also written using the Python language and they can be run as such in both Windows and Linux environments. Also the Subscribe-client was implemented for mobile and tablet users using Android and Java language.

Keywords: web-programming, Linux, Windows, Android, C#, Python, Java, open source code, Publish-Subscribe pattern, Mono

SISÄLTÖ

Kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

Mono Avoimen lähdekoodin .NET-framework, jolla voidaan esimerkiksi ajaa Microsoftin työkaluilla käännettyä C#-binäärikoodia Linux-ympäristössä (Mono [viitattu 23.10.2014]).

Rasberry Pi Luottokorttikokoinen tietokone, jossa voidaan ajaa Linux-käyttöjärjestelmää ja johon on helppo kytkeä ulkoisia antureita erilaisilla väylillä (Raspberry Pi Foundation [viitattu 23.10.2014]).

UDP TCP/IP-protokollaperheen protokolla, joka tarjoaa yhteydettömän tietosanomien kuljetuspalvelun (Comer 2002, 198).

Socket Socket on käyttöjärjestelmämekanismi, joka muodostaa kommunikoinnin päätepisteen. Sovellusohjelma voi pyytää käyttöjärjestelmää luomaan Socketin ja sitoa sen käytettyyn protokollaan ja yhteystyyppiin. (Comer 2002, 415.)

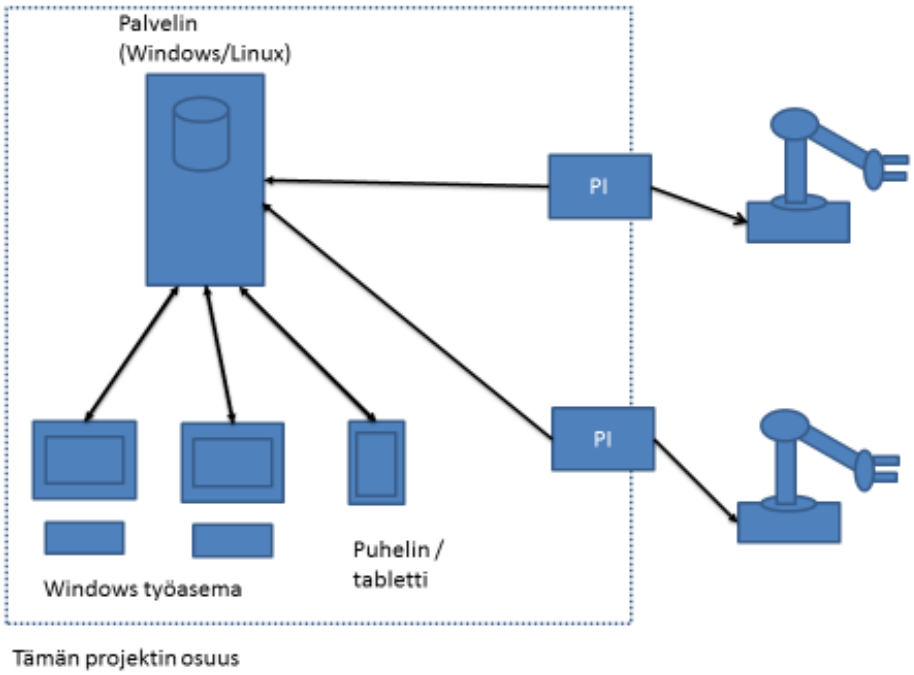
Python Tulkattava ohjelmointikieli, jolle löytyy toteutus sekä Windows, että Linux-järjestelmistä.

TCP/IP TCP/IP on viralliselta nimeltään TCP/IP Internet Protocol Suite -protokollaperheestä yleisesti käytetty lyhenne. Se määrittelee kuinka laitteet kommunikoivat internetin yli toistensa kanssa ja miten tieto niiden välillä lähetetään. (Comer 2002, 2.)

1 JOHDANTO

1.1 Työn tausta

Raspberry Pi -kone on luottokortin kokoinen yhden piirilevyn tietokone, johon on mahdollista helposti kytkeä erilaisia antureita, kerätä näillä tietoa lämpötiloista, paineesta, pinnankorkeudesta tai mitata jännitettä esimerkiksi aurinkopaneelista. Raspberry Pi -konetta voidaan myös käyttää lukemaan terminaaliliitännän avulla robottien normaalisti paikalliselle päätteelle tulostamia lokeja. Hankittu tieto on kuitenkin tarpeen välittää eteenpäin jatkokäyttöä varten varsinaisille asiakasohjelmille. Näitä tietoja voidaan tarpeen mukaan esittää käyttäjälle Windows-työasemilla, älypuhelimilla tai tableteilla. Seuraavassa kuviossa 1 on esitetty yksi mahdollinen käyttöympäristö, jossa robottien terminaali liitännästä saatavat lokit välitetään käyttäjille.



Kuvio 1. Esimerkkijärjestelmä

Järjestelmässä robottiin kytketty Raspberry PI -kone lukee lokitietoa robotin sarjaportista. Luettuaan lokit kone lähettää ne palvelimelle. Palvelin tallentaa vastaanot-

tamansa lokit tietokantaan ja välittää ne reaaliaikaisesti palvelimelle rekisteröityneille asiakkaille. Palvelimelle rekisteröityessään asiakas saa ensin palvelimella tallessa olevan historiatiedon ja sen jälkeen kaikki uudet lokitapahtumat reaaliaikaisesti.

Tässä opinnäytetyössä keskitytään tiedon keruun sijaan tutkimaan miten kerätyn tiedon reaaliaikaiseen välittäminen asiakasohjelmille olisi mahdollista toteuttaa. Toteutusta suunniteltaessa on lähdetty siitä, että tiedot on saatava välitettyä myös eri käyttöjärjestelmien välillä, eli työasemat voivat olla joko Linux- tai Windows-työasemia. Puhelimissa ja tableteissa käytössä voisi olla Android- tai Windows-käyttöjärjestelmä.

1.2 Työn tavoitteet

Opinnäytetyön tavoitteena on rakentaa Publish-Subscribe-ohjelmistomallin mukainen järjestelmä, jossa Rasberry Pi -koneen avulla kerätty tieto välitetään asiakasohjelmille. Ohjelmistomallin Publish-asiakasta ajetaan Raspberry Pi -koneessa ja Subscribe-asiakasta Windows-työasemassa. Itse palvelin, joka toimii tiedon jakelijana asiakkaiden välillä, voi olla joko Windows- tai Linux-palvelimella. Tapahtumien jakelun lisäksi palvelin ylläpitää historiatietoa, joka ladataan Subscribe-asiakkaalle sen käynnistyessä.

Työn toisena tavoitteena on ollut testata avoimen lähdekoodin .NET-toteutusta, joka mahdollistaa C#-binäärikoodin ajamisen sellaisenaan Linux-koneilla.

Kolmantena tavoitteena on tutustua UDP-socketeihin perustuvaan viestin välitykseen eri käyttöjärjestelmien välillä. Tässä yhteydessä toteutettiin Python-kieliset versiot asiakasohjelmista sekä toteutettiin Subscriber-asiakas Android-puhelimelle ja -tabletille.

1.3 Työn rakenne

Luku 2 käsittelee ohjelmistomalleja. Siinä esitellään tässä työssä käytetty Publish-Subscribe-ohjelmistomalli ja sen perustana oleva asiakas-palvelin-malli.

Luvussa 3 käsitellään TCP/IP-tietoverkkoprotokollaperheen UDP-protokolla, ja sen Socket-ohjelmistorajapinta. Lisäksi siinä käsitellään C#- , Python- ja Java-kielen tarjoamia tapoja käyttää socket-rajapintaa.

Luvussa 4 esitelty Linuxissa käytetty MONO .NET framework -toteutus ja Raspberry Pi -tietokone, jossa asiakasohjelmia ajetaan.

Luku 5 kuvaa toteutetun järjestelmän vaatimukset loogisella tasolla käyttötapauksina kuvattuna.

Luvussa 6 käydään läpi käytetyt kehitys- ja ajoympäristöt ja kuvataan varsinainen toteutus luokkakaavion, käyttötapausten sekvenssikaavioiden avulla.

Luvussa 7 esitellään työn tulokset ja yhteenveto.

2 OHJELMISTOMALLI

2.1 Ohjelmistomallin perusajatus

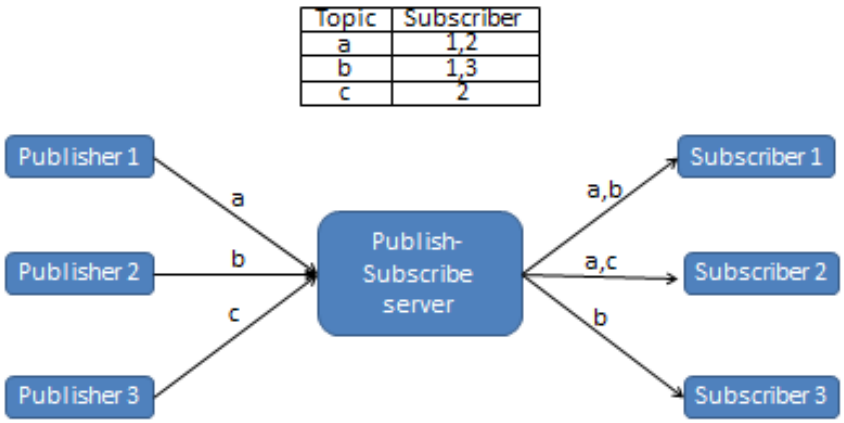
Ohjelmistokehityksessä tulee usein vastaan samankaltaisia ongelmia, joihin voi soveltaa samoja ratkaisumalleja. Näitä valmiita ohjelmistomalleja on kuvattu kirjoissa ja internetissä. Malleja on olemassa monen tasoisia alkaen järjestelmätasolta päätyen aina algoritmien toteutusmalleihin. Tässä työssä on käytetty asiakas-palvelin-malliin perustuvaa Publish-Subscribe-ohjelmistomallia, joka sopii tiedon välittämiseen asiakkaalta toisille asiakkaille. Näitä malleja on kuvattu tarkemmin seuraavissa kappaleissa.

2.2 Asiakas-palvelin-malli

Asiakas-palvelin-malli koostuu kahdesta ohjelmasta, jotka välittävät toisilleen tietoa verkon kautta. Termi palvelin kuvaa ohjelmaa, joka tarjoaa verkon kautta käytettävän palvelun. Se vastaanottaa verkosta tulevan palvelupyynnön, toteuttaa sen ja palauttaa tuloksen asiakkaalle. Asiakas on ohjelma, joka lähettää pyynnön palvelimelle ja odottaa sen vastausta. (Comer 2002, 403-404.)

2.3 Publish-Subscribe-ohjelmistomalli

Kuviossa 2 esitellään aiheen tilaajalistaan perustuva Publish-Subscribe-ohjelmisto-malli. Tässä mallin toteutuksessa palvelin ylläpitää taulukkoa, jossa on tallennettuna aihekohtaiseen listaan kaikki asiakkaat, jotka ovat kiinnostuneet tietystä aiheesta. Kun Publisher-asiakas lähettää palvelimelle uuden tapahtuman Publish-palvelulla, palvelin käy hakemasta listastaan kaikki tästä aiheesta kiinnostuneet Subcriber-asiakkaat ja välittää tämän tapahtuman kaikille siitä kiinnostuneille. Subscriber-asiakas rekisteröityy palvelimen listalle Subscribe-palvelua käyttäen ja myös poistaa itsensä listalta kutsumalla UnSubscribe-palvelua, kun ei ole enää tapahtumista kiinnostunut. (Code Project 2009.)



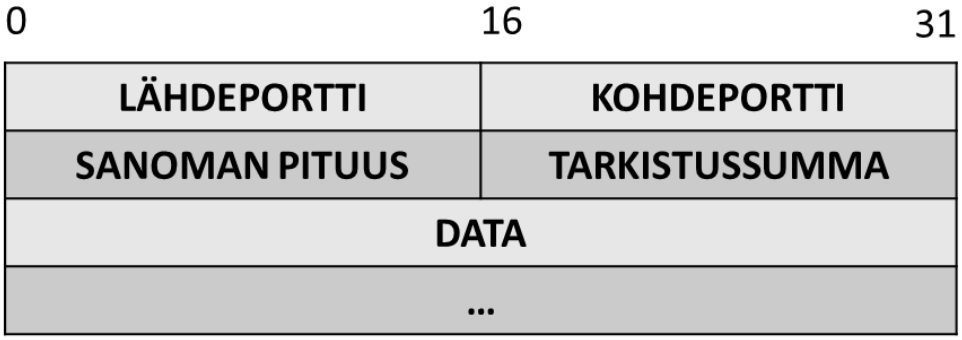
Kuvio 2. Tilaajalistaan perustuva Publish-Subscribe-ohjelmistomalli (Code Project 2009).

3 UDP-PROTOKOLLA JA SEN KÄYTTÖ

3.1 UDP-protokolla

UDP-protokolla on TCP/IP-perheen protokolla, joka tarjoaa yhteydettömän tavan välittää tietoa laitteelta toiselle. UDP käyttää tietosähkeitä lähettäessään IP-protokollaa. Se ei tarkista sanomien perillemenoa lähettämällä kuittauksia. UDP ei myöskään takaa saapuvien sanomien järjestystä, eli vastaanotetut sanomat voivat saapua määränpäähän eri järjestyksessä kuin ne on lähetetty. UDP:tä käyttävän sovelluksen on itse huolehdittava kuittauksista ja järjestettävä sanomat vastaanottopäässä uudelleen tarvittaessa. (Comer 2002, 198-199.)

Kuviossa 3 on esitelty UDP-tietosähkeen sisältö. UDP-tietosähke sisältää datan lisäksi lähde- ja kohdeportit. Ne sisältävät 16-bittisen portin numeron, jonka perusteella protokollaohjelmisto välittää datan sovellusprosessille. Lähdeportti on valinnainen. Jos lähdeportti on määritelty, se sisältää portin johon sovellusohjelma lähettää vastauksensa. (Comer 2002, 199.)



Kuvio 3. UDP-tietosähkeen kentät (Comer 2002, 199).

3.2 Socket

Socket-ohjelmistorajapinta tarjoa käsitteen Socket. Se on käyttöjärjestelmän palvelu, jota käyttäen voidaan vastaanottaa tai lähettää dataa IP-yhteyttä käyttäen. Yhteyden toisessa päässä käytetään write-kutsua tiedon lähettämiseen ja toisessa päässä read-kutsua tiedon vastaanottamiseen. (Comer 2002, 415.)

Socketit luodaan tarvittaessa new-funktiolla. Socketille annetaan alustusparametreina protokollaperhe, joka määrittää kuinka osoitteet tulkitaan. Lisäksi määritellään yhteyden tyyppi sekä käytettävä protokolla. Tässä tapauksessa käytettäväksi protokollaksi valitaan UDP-protokolla, jolloin yhteys täytyy valita yhteydettömäksi tietosähkeiden kuljetuspalveluksi. Kun Socketin käyttö päättyy, täytyy se sulkea close-metodilla. (Comer 2002, 415-416.)

Jotta luotua sockettia voidaan käyttää tietosähkeiden välittämiseen, täytyy se vielä sitoa paikaaliseen osoitteeseen palvelimessa. Vastaavasti asiakassovelluksessa käytetään palvelimen osoitetta ja porttia kohdeosoitteena (Comer 2002, 417-419). Kun Socket on luotu ja sidottu, voidaan yhteydettömiä tietosähkeitä lähettää SendTo-metodilla ja vastaanottaa ReceiveFrom-metodilla. (Comer 2002, 420,422).

3.3 Windowsin Socket-luokka

Microsoftin .NET Framework 4.5 -ohjelmistokehys tarjoaa Socket-luokan, joka toteuttaa edellisessä kappaleessa kuvatun Socket-ohjelmistorajapinnan käytettäväksi C#-ohjelmista käsin (Microsoft [viitattu 19.11.2014]).

3.4 Pythonin Socket-kirjasto

Python-ohjelmointikielessä on valmis Socket-kirjasto, joka tarjoaa mahdollisuuden käyttää Socketteja vastaavaan tapaan kuin C#-kielen Socket-kirjastot. Ensin luodaan Socketti-olio. Socketti-olion alustusparametreillä määritellään käytettävä protokollaperhe ja yhteyden tyyppi. Sen jälkeen Socket-olio sidotaan käytettyyn osoitteeseen ja porttiin. Tämän jälkeen datan lähetykseen ja vastaan ottamiseen ovat käytettävissä vastaavat sendto- ja recvfrom-metodit kuin C#-kielessä. (BinaryTides 2012.)

3.5 Androidin java.net-kirjasto

Android tarjoaa Sockettien käyttöön standardit java.net-kirjastot. Tämä kirjasto tarjoaa monia valmiita luokkia verkkorajapinnan toteutukseen. Socket-asiakastoteutukseen tarvitaan kahta luokkaa: DatagramSocket ja DatagramPacket. Ensin luodaan DatagramSocket-olio. Tämän jälkeen lähetettävä sanoma pakataan DatagramPacket-olion rakentajalla, jolle kerrotaan sanoma, sen pituus, vastaanottajan IP-osoite ja portti. Sen jälkeen sanoma lähetetään käyttäen DatagramSocket-instanssin send-metodia. Sanoman vastaanotto tapahtuu vastaavasti luomalla ensin DatagramPacket-olio, joka annetaan parametrina DatagramSocket-olion receive-metodille. (Oracle [viitattu 10.2.2015].)

4 KÄYTETYT OHJELMISTOT ja LAITTEET

4.1 Mono

Mono on avoimeen lähdekoodiin perustuva .NET-framework -ohjelmisto, joka sallii alustariippumattoman ohjelmistokehityksen ja ohjelmien ajamisen. Mono-toteutus perustuu ECMA-standardeihin C# (ECMA 2006) ja Common Language Infrastrure (ECMA 2012). (Mono [viitattu 7.11.2014].)

Toteutus koostuu C#-kääntäjästä, Microsoftin .NET -kirjastojen kanssa yhteensopivista kirjastoista ja muista apukirjastoista sekä ajoympäristöstä. Se on yhteensopiva Microsoftin API -rajapintojen kanssa ja ajaa ASP.NET, ADO.NET, Silverlight sekä Windows.Forms -ohjelmia uudelleen kääntämättä, siis suoraan Microsoftin työkaluilla käännettyä binäärikoodia. (Mono [viitattu 7.11.2014].)

Tässä työssä Mono-ohjelma asennettiin Raspberry Pi -koneeseen ja sitä käytettiin Microsoftin työkaluin tuotetun C#-binäärikoodin ajamiseen Linux-koneella.

4.2 Raspberry Pi -tietokone

Raspberry Pi on halpa luottokorttin kokoinen tietokone. Kuviossa 4 on kuva Raspberry Piin versio 1 B -mallista. Raspberry-koneeseen voi kytkeä ulkoisia antureita esimerkiksi käyttäen sen tarjoamaa IC2-väylää. Kappaleessa 1.1 kuvatussa esimerkkijärjestelmässä koneella kytkeydytään ohjelmallisesti toisen laitteen sarjaväylään ja luetaan sitä ohjelmassa. Tätä kirjoitettaessa Raspberry Pi -koneesta on jo saatavilla versio 2, joka on yhteensopiva tätä työtä tehdessä käytetyn version 1 kanssa. (Raspberry Pi Foundation. [viitattu 19.11.2014].)

Kuvio 4. Raspberry Pi 1 model B

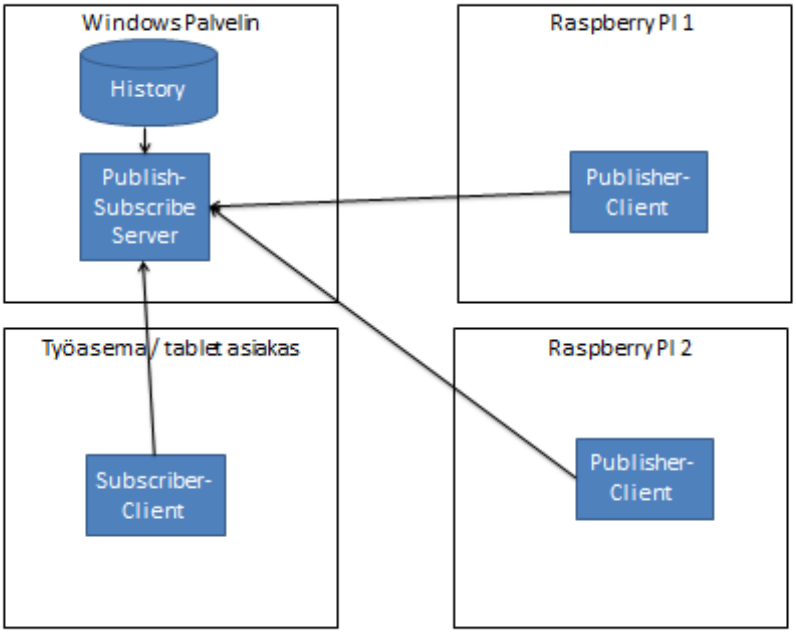
Raspberry Pi -koneessa voidaan käyttää useita eri käyttöjärjestelmän jakeluja. Tässä työssä Raspberry Pi -koneeseen ladattiin ja asennettiin Raspberry Pi foundatio -sivustolta Debian Wheezy Linux –jakelu. (Raspberry Pi Foundation. [viitattu 19.11.2014].)

Raspberry Pi -koneella ajettiin asiakassovelluksia suoraan C#-binäärikoodia käyttäen edellisessä kappaleessa kuvattua Mono-ohjelmaa. Myös Python-koodina toteutettuja asiakassovelluksia testattiin ajamalla niitä Raspberry Pi -koneesta käsin.

5 TIEDONVÄLITYSJÄRJESTELMÄ

5.1 Tiedonvälitysjärjestelmän kuvaus

Tiedonvälitysjärjestelmän toteutus koostuu asiakasohjelmista ja palvelinohjelmasta. Kuviossa 5 on kuvattu ohjelmiston komponenttien jakautumien eri laitteisiin. Asiakasohjelmista Publisher-asiakasta ajetaan tietoa keräävässä Raspberry Pi -koneessa ja Subscriber-asiakasta joko Windows-työasemissa tai Android-älypuhelimissa tai -tableteissa. Palvelinkoneessa ajetaan palvelinohjelmaa, joka pitää tallessa myös historiatietokantaa.

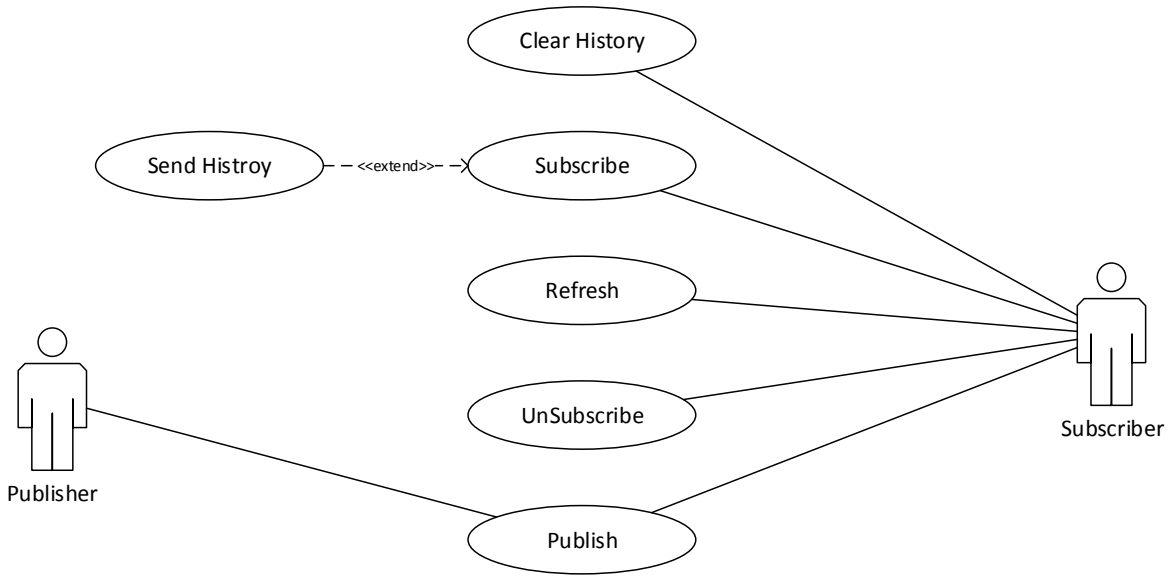


Kuvio 5. Asiakas- ja palvelinohjelmien jako eri laitteisiin

5.2 Käyttötapaukset

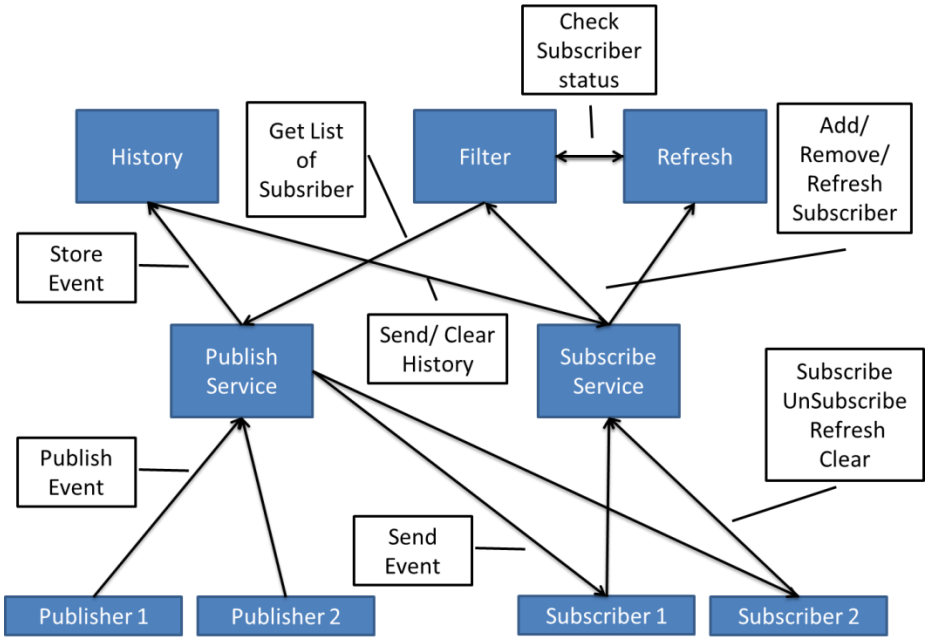
Publish-Subscribe-ohjelmistomalli pitää sisällään kolme pääkäyttötapausta: Publish, Susbcribe ja UnSubscribe. Näiden käyttötapausten lisäksi on tehty oma Refresh-käyttötapaus asiakaslistan virkistykseen, sekä oma Clear History -käyttötapaus historia tietojen poistamiseen palvelimelta. Asiakaslistan tarkistus on lisätty

Publish-käyttötapaukseen ja historiatiedon lähetys asiakkaalle Subscribe-käyttötapaukseen. Käyttötapaukset ja niiden toimijat on kuvattu kuviossa 6.



Kuvio 6. Tiedonkeräysjärjestelmän käyttötapaukset

Käyttötapausten kuvausta varten palvelimen toiminnallisuus on jaettu viiteen osaan. Kuviossa 7 on kuvattu nämä viisi osaa ja niiden väliset tapahtumat. Kuviossa on mukana myös asiakasohjelmat ja niiltä tulevat pyynnöt.



Kuvio 7. Palvelimen toimintojen looginen jako käyttötapauskuvauksissa käytettyihin luokkiin

Publish-palvelu, Subscribe-palvelu ja Filter-luokka toteuttavat ohjelmistomallin mukaisen Publish-Subscribe-palvelun, jossa Filter-luokan vastuulla on pitää kirjaa aiheista kiinnostuneista Subscriber-asiakkaista.

Lisäksi perusohjelmistomalliin on lisätty toiminnallisuus, jossa palvelin pitää yllä tietokantaa tapahtumahistoriasta. Se tallentaa Publish-käyttötapauksessa tapahtumat History-luokan ylläpitämään tietokantaan. Subscribe-käyttötapausta on laajennettu siten, että palvelin lähettää tapahtumahistorian asiakasohjelmalle, kun asiakas tekee tilauksen tietystä aiheesta. Täysin uusi Clear-käyttötapaus on määritelty tilanteisiin, joissa asiakasohjelmisto haluaa tyhjentää palvelimen tietokannasta tietyn aiheen historiatiedot.

Koska toteutus päätettiin tehdä yhteydettömällä UDP-protokollalla, sanomien perillemenoa ei kuitata mitenkään protokollassa. Tämän vuoksi lisättiin perusohjelmis-

tomalliin toiminnallisuus, jossa Susbcriber-asiakas kertoo tilausta tehdessään kuinka monen tapahtuman välein se ilmoittautuu uudelleen Susbcribe-palvelulle. Ilmoittautumista varten on lisätty oman Refresh-käyttötapaus. Publish-käyttötapausta on laajennettu siten, että se poistaa Subscriber-asiakkaan tilauksen, jos asiakas virhetilanteessa jättää virkistämisen tekemättä. Tällä lisäyksellä saadaan palvelimen asiakaslista pidettyä ajantasaisena.

5.2.1 Publish

Publish-käyttötapauksessa Publisher-asiakas lähettää tietylle aiheelle tapahtuman, jonka palvelin välittää aiheesta kiinnostuneille Subscriber-asiakkaille.

Ennakkoehtona on:

- Joku Subscriber-asiakas on tehnyt tilauksen sisään tulevan sanoman aiheelle.

Normaali toiminta on:

1. Publisher-asiakas lähettää Publish-pyynnön.

   Publish,<time>,<topic>,<eventdata>

2. Publish-palvelu ottaa tapahtuman vastaan ja tallentaa sen historiaansa.

3. Publish-palvelu tarkistaa Filter-luokalta kuka on tilannut aiheen mukaiset tapahtumat.

4. Filter-luokka etsii sisältä listaltaan aiheen tilanneet asiakkaat. (E4)

5. Filter-luokka tarkistaa Refresh-luokalta että kaikki asiakkaat ovat vielä aktiivisia, ja palauttaa tarkistetun listan Publisher-palvelulle. (E5)

6. Publish-palvelu lähettää tapahtuman eteenpäin Subscribe-asiakkaille asiakaslistan mukaan.

Poikkeukset ovat:

- E4: Filter-luokalla ei ole tilaajia aiheelle, tapahtuma tallennetaan vain Historia-luokan ylläpitämään tietokantaan.

- E5: Kaikki Filter-luokan tilaajat ovat vanhentuneita, tapahtuma tallennetaan vain historiatietokantaan.

Lopputulos on:

- Tapahtuma on välitetty kaikille siitä kiinnostuneille aktiivisille Subscriber-asiakkaille.

5.2.2 Subscribe

Subscribe-käyttötapauksessa Subscriber-asiakas tekee tietylle aiheelle tapahtumatilauksen palvelimelle.

Ennakkoehtona on:

- Subscriber-asiakas ei ole aiemmin tilannut aiheen mukaisia sanomia.

Normaali toiminta on:

1. Subscriber-asiakas lähettää Subscribe-pyynnön.

   Subscribe, <topic>,<refreshlimit>

2. Subscribe-palvelu ottaa tapahtuman vastaan ja lisää Subscriber-asiakkaan Filter-luokan ja Refresh-luokan listoille. (E2)

3. Subscribe-palvelu pyytää History-luokkaa lähettämään mahdolliset tallennetut tapahtumat asiakkaalle. (E3)

Poikkeukset ovat:

- E2: Jos asiakas löytyy jo listoilta, ei sitä tallenneta toiseen kertaan vaan päivitetään tilauksen olemassa oleva refershlimit-arvo.

- E3: Jos historia on tyhjä, niin mitään tapahtumia ei lähetetä tilauksen tehneelle asiakkaalle.

Lopputulos on:

- Subscriber-asiakas on tilannut aiheen mukaiset tapahtumat ja on aktiivinen.
- Subscriber-asiakkalle on lähetetty mahdollinen tapahtumahistoria.

5.2.3 UnSubscribe

UnSubscribe-käyttötapauksessa Subscriber-asiakas poistaa tietyn aiheen tapahtumatilauksensa palvelimelta.

Ennakkoehtona on:

- Subscribe-asiakas on tilannut aiheen mukaiset tapahtumat.

Normaali toiminta on:

1. Subscriber-asiakas lähettää UnSubscribe-pyynnön.

   UnSubscribe, <topic>

2. Subscribe-palvelu ottaa tapahtuman vastaan ja poistaa Subscriber-asiakkaan Filter-luokan ja Refresh-luokan listoilta. (E2)

Poikkeuksena on:

- E2: Jos asiakasta ei löydy aiemmin tilauksen tehneiden listoilta, pyyntö hylätään hiljaisesti.

Lopputulos on:

- Subscriber-asiakkaan tilaus aiheelle on peruttu.

5.2.4 Refresh

Refresh-käyttötapauksessa Subscriber-asiakas päivittää tietyn aiheen tapahtuma-tilauksensa palvelimella.

Ennakkoehtona on:

- Subscribe-asiakas on tilannut aiheen mukaiset tapahtumat.

Normaali toiminta on:

1. Subscriber-asiakas lähettää Refresh-pyynnön.

   Refresh, <topic>,<refreshlimit>

2. Subscribe-palvelu ottaa tapahtuman vastaan ja päivittää Subscriber-asiakkaan aktiivisuustiedon Refresh-luokan ylläpitämälle listalle. (E2)

Poikkeuksena on:

- E2: Jos asiakasta ei löydy aiemmin tilauksen tehneiden listoilta, pyyntö hylätään hiljaisesti.

Lopputulos on:

- Subscriber-asiakaan tapahtumatilaus on päivitetty ja aktiivinen.

5.2.5 Clear

Clear-käyttötapauksessa Subscriber-asiakas poistaa tietyn aiheen historiatiedot palvelimelta.

Ennakkoehtona on:

- Aiheelle löytyy historiatietoja.

Normaali toiminta on:

1. Subscriber-asiakas lähettää Clear-pyynnön.

Clear, <topic>

2. Subscribe-palvelu ottaa tapahtuman vastaan ja pyytää History-luokkaan poistamaan aiheen historian. (E2)

Poikkeuksena on:

- E2: Jos aiheelle ei löydy historiaa, pyyntö hylätään hiljaisesti.

Lopputulos on:

- Aiheen historiatiedot on tyhjennetty.

6 TYÖN TOTEUTUS

6.1 Kehitys- ja ajoympäristö

Toteutusta tehdessä käytettiin kehitysympäristönä Windows-työasemaa, johon on asennettu Microsoftin Visual Studio C#-koodin tuottamista ja kääntämistä varten. Työaseman kanssa samaan verkkoon oli kytketty Raspberry Pi -kone, jossa käytettiin Debian Wheezy Linux -jakelua. Raspberry Pi -koneeseen asennettiin myös Python-ajoympäristö sekä MONO-framework-ohjelmisto Windows-työasemassa käännettyjen C#-ohjelmien ajamiseksi.

Android-sovelluksen toteutukseen ja testaukseen käytettiin samaan Windows-työasemaan asennettua Android Studio -ohjelmistoa. Ohjelmisto on ladattavissa vapaasti Androidin Developer-sivustolta (Android [viitattu 10.2.2015]). Samalla ohjelmistolla on mahdollista testata omaa ohjelmaa ajamalla sitä joko emulaattorissa tai varsinaisessa Android-päätelaitteessa, joka on kytketty kehityskoneeseen USB-kaapellilla.

Myös testaus tehtiin ajamalla palvelinohjemaa samalla kehitysympäristönä käytetyllä Windows-työasemalla. Työaseman palomuuri konfiguroitiin sallimaan UDP-yhteydet samassa verkossa olevalta Raspberry Pi -koneelta, toiselta Windows-työasemalta ja Android-tabletilta. Näissä muissa verkossa olevissa laitteissa ajettiin sitten asiakasohjelmia.

Seuraavissa kappaleissa kuvataan tarkemmin Raspberry Pi -tietokoneen alustus ja konfigurointi. Näiden toimenpiteiden jälkeen sinne voidaan SFTP:llä siirtää ajettava C#-ohjelma tai Python-skripti. Siirron jälkeen C#-ohjelmaa voidaan ajaa MONO-ohjelmistolla ja skriptiä Python-tulkilla.

6.1.1 Debian Wheezyn asennus Rasbian Pi -koneeseen

Debian Wheezy -jakelu asennettiin Rasbian Pi -koneeseen Raspberry Pi foundationin -sivuston ohjeiden mukaan (Raspberry Pi Foundation. [viitattu 6.2.2015]). Ensin ladattiin 2015-01-31-rasbian-image PC:lle organisaation tarjoamalta web-

sivulta (Raspberry Pi Foundation. [viitattu 7.2.2015]). Tämä ladattu image siirrettiin sitten ohjeiden mukaan SD-kortille Win32DiskImager-ohjelmalla (Raspberry Pi Foundation. [8.2.2015]).

Ensimmäisen käynnistyksen yhteydessä Raspberry Pi -kone käynnistyy konfigurointivalikkoon. Tästä valikosta säädettiin muutamaa asetusta.

- Expand FileSystem -valikosta vapautetaan kaikki tila SD-kortilta käyttöön.
- Change Locale -valikosta valittiin käyttöön merkistöksi fi_FI.UTF-8.
- ChangeTimezone -valikosta valittiin aikavyöhykkeeksi Helsinki.

Tämän jälkeen Raspberry Pi -kone on hyvä päivittää ajan tasalle seuraavalla komennolla:

sudo apt-get update

SSH- ja SFTP-yhteydet toimivat tämän jälkeen oletusasetuksilla. Myös Python-tulkki on mukana tässä uusimmassa imagessa.

6.1.2 MONOn asennus Rasbian Pi-koneeseen

Tässä työssä käytettävä Mono-framework -ohjelmisto asennettiin Rasbian Pi -koneen Debian Wheezy -jakelun päälle. Itse asennus onnistuu seuraavalla pakettinhallinnan komennolla:

sudo apt-get install mono-complete

Samalla asentuu myös C#-kääntäjä *gmcs,* jota käyttäen on helppo todentaa asennuksen onnistuminen kirjoittamalla editorilla lyhyt ”Hello world” -ohjelma.

```
using System;
 public class HelloWorld
{
   public static void Main()
   {
```

```
            Console.WriteLine("Hello World!");
        }
    }
```

Ohjelman tallennuksen jälkeen se voidaan kääntää ja ajaa se.

```
gmcs HelloWorld.cs
mono HelloWorld.exe
```

(Dan's Website [viitattu 24.10.2014]).

Kun Mono-framework on asennettu Rasberry Pi:lle, Windows-ympäristössä Microsoftin työkaluin käännetyt C#-asiakasohjelmat ovat ajettavissa siirron jälkeen suoraan seuraavilla komennoilla:

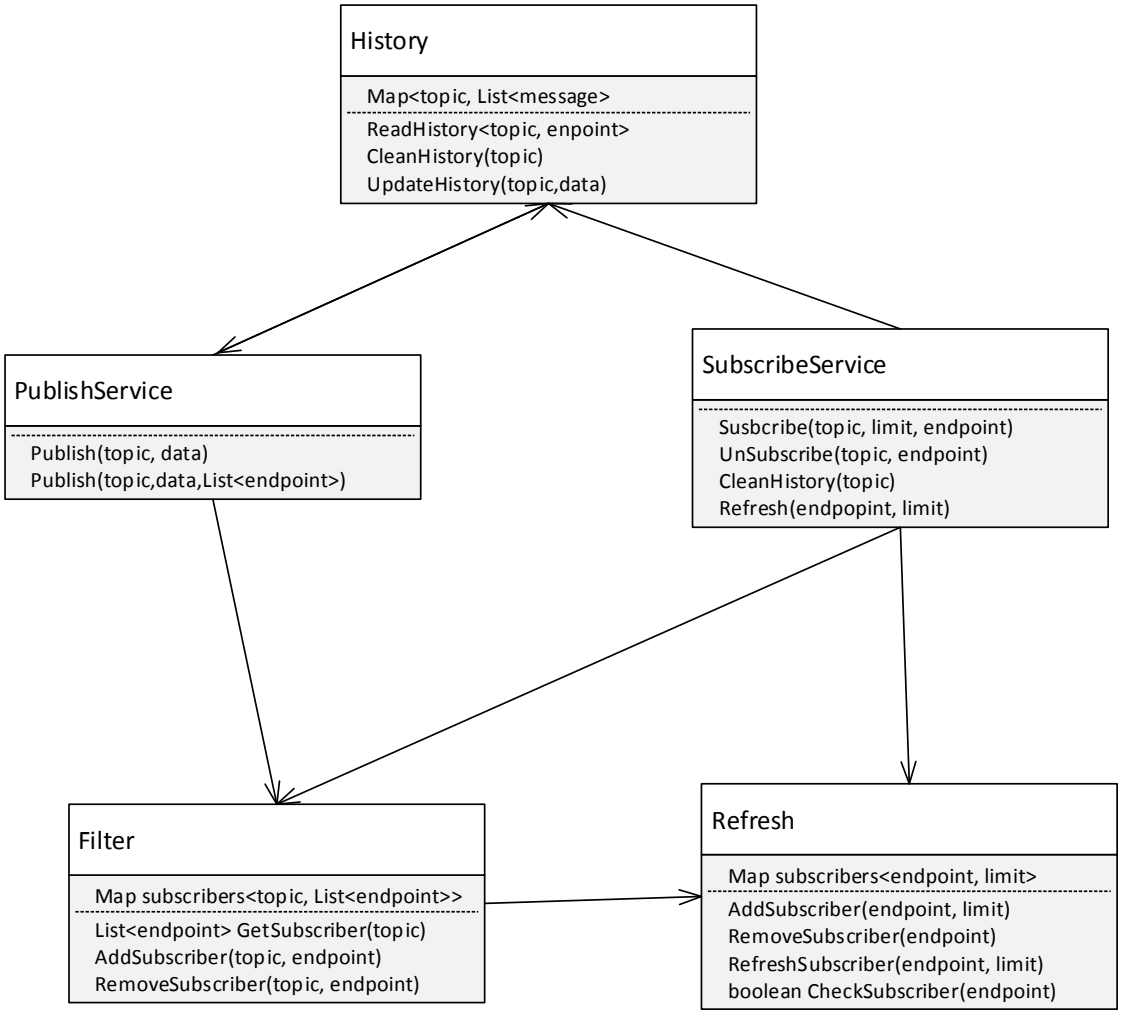
```
mono SocketBasedPublisherConsole.exe <ip> <topic> <eventdata>

mono SocketBasedSubscriberConsole.exe <ip> <topic>
```

Kun palvelua ajettiin Windows-koneessa, oli Windows-palomuuri konfiguroitava sallimaan liikenne UDP-protokollaa käyttäen Raspberry Pi -koneen suuntaan. Koska Subscribe-asiakas voi käyttää mitä tahansa porttia paluukanavanaan, ei palomuurisääntöihin voi tehdä IP-osoitetta tarkempia rajauksia UDP-protokollan käytöstä.

6.2 Käyttötapausten toteutukset

Tässä kappaleessa on kuvattu kappaleessa 5.2 kuvattujen käyttötapausten suunniteltu toteutus sanomakaavioin. Kuviossa 8 on kuvattu toteutuksessa käytettäväksi suunnitellut luokat. Luokkiin on kuvattu myös operaatiot, joita käytetään sanomakaavioissa. Luokkien attribuutteina on määritelty listat, joissa säilötään tapahtuman aihe ja asiakaskohtaisia tietoja. History-luokan vastuulla on historiatiedon ylläpito. PublishService-luokka toteuttaa nimensä mukaan välityspalvelun asiakkaille, eli ottaa vastaan viestit Publish-asiakkailta ja välittää ne edelleen tilaajille. SubscribeService-luokka toteuttaa tilauspalvelun Susbcriber-asiakkaalle. SubscriberService-luokka toteuttaa tilaustoiminteet: tilauksen tekeminen, tilauksen peru-

minen ja tilauksen päivittäminen. Lisäksi SubscriberService-luokka tarjoaa palvelun historiatiedon tyhjentämiseen. Filter-luokka ylläpitää aihekohtaista asiakaslistaa ja palauttaa listan pyydettäessä. Publish-palvelu pyytää Filter-luokalta asiakaslistan, jolle se välittää tietyn tapahtuman. Refesh-luokka ylläpitää asiakaskohtaista tietoa siitä, koska sovittu määrä tapahtumia on välitetty eteenpäin. Refesh-luokka päivittää listaansa, kun se saa Subscribe-palvelun kautta asiakkaalta Refresh-pyynnöllä tiedon, että asiakas on edelleen aktiivinen ja odottaa lisää sanomia.

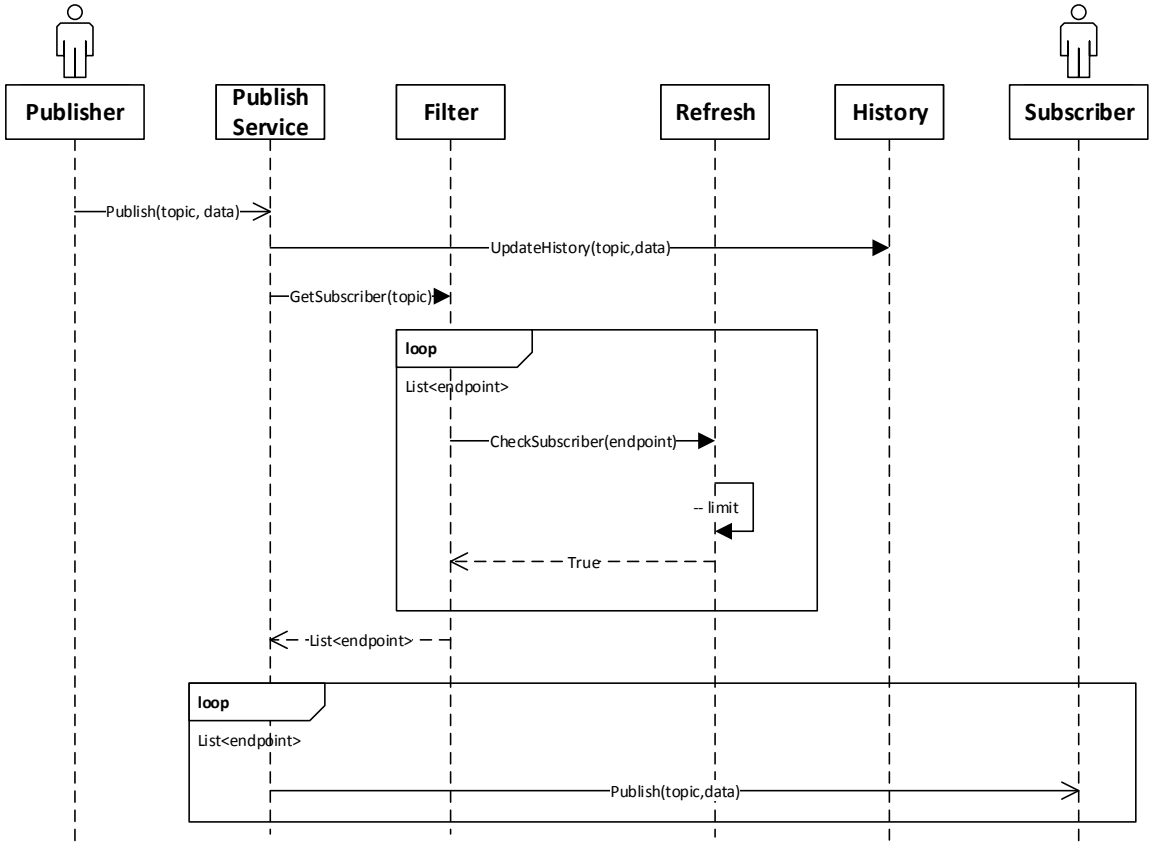


Kuvio 8. Toteutussuunnitelman luokkajako

6.2.1 Publish

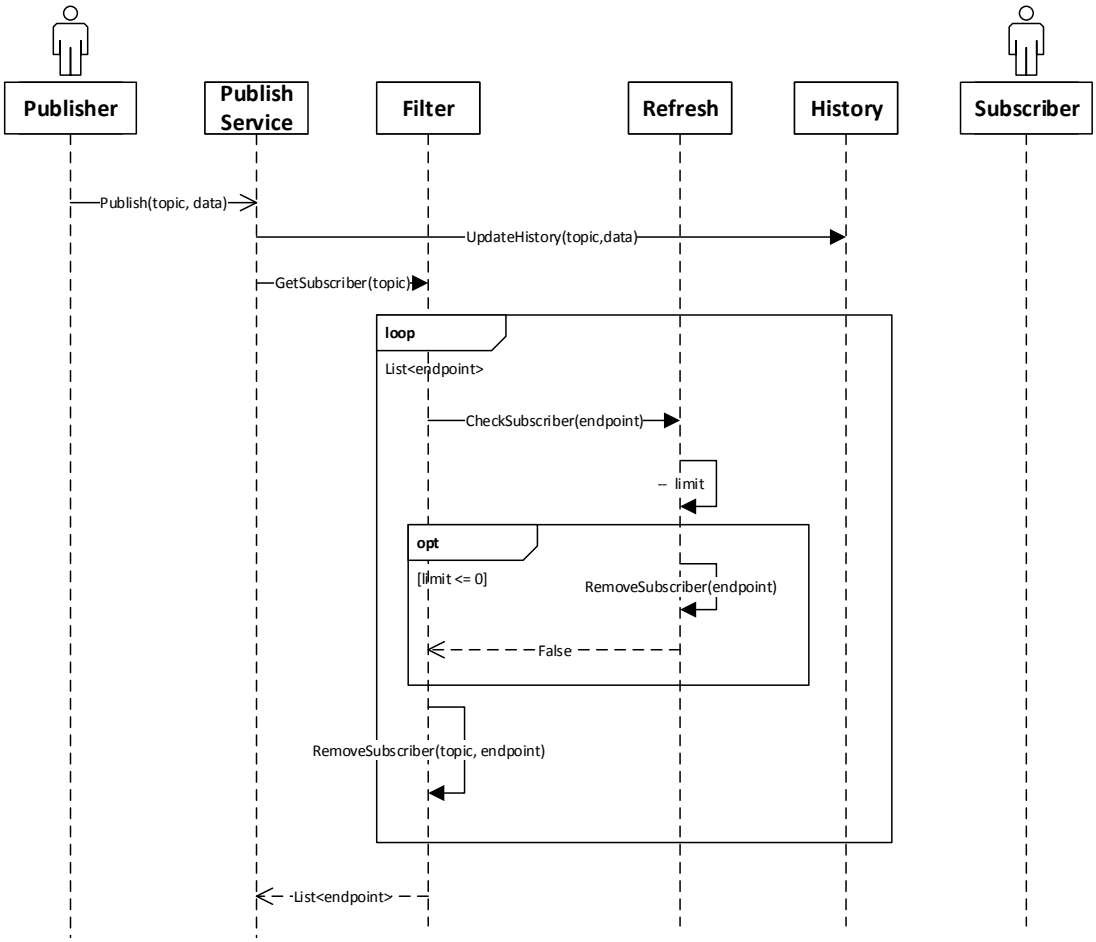
Publish-pyynnön käsittely alkaa tapahtuman tallentamisella historiakantaan. Seuraavaksi palvelu selvittää tapahtumasta kiinnostuneet tilaajat. Samalla tarkistetaan myös että tilaaja on vielä aktiivinen. Lopuksi tapahtuma välitetään kaikille tämän

aiheen tilanneille aktiivisille asiakkaille. Kuviossa 9 käyttötapaus on kuvattu sekvenssikaaviona.



Kuvio 9. Viestin välitys tilaajille

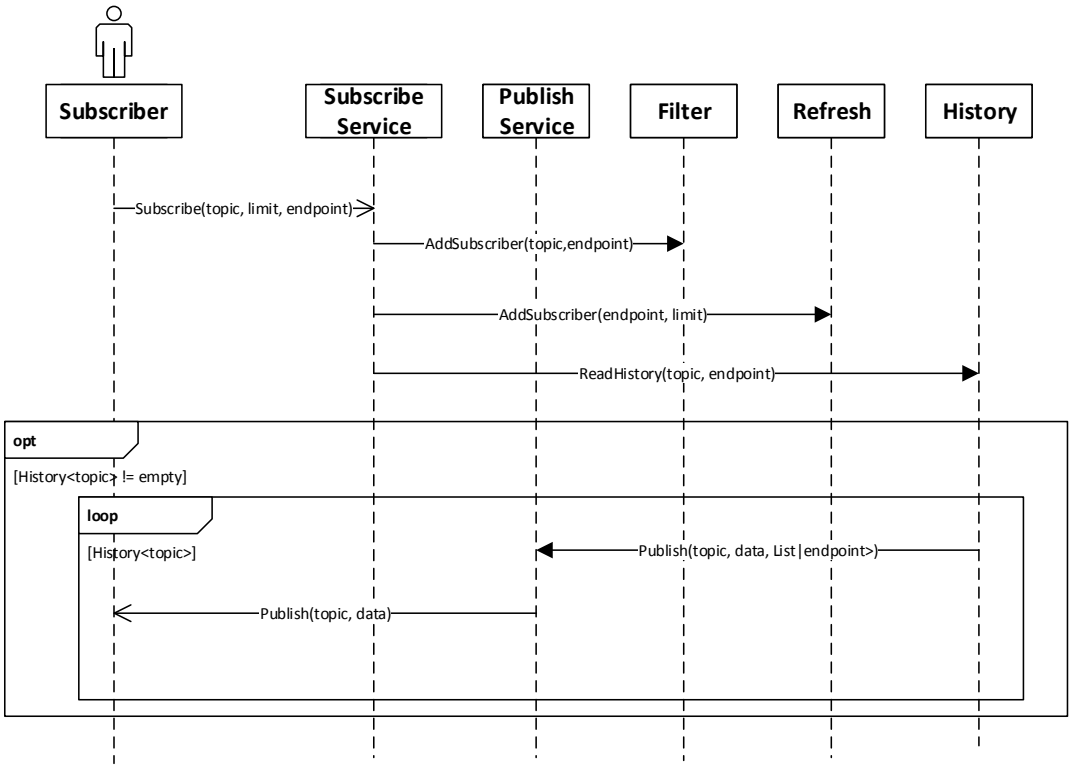
Jos tilaajan tilaa tarkistettaessa huomataan, että tilaaja ei ole enää aktiivinen, poistetaan tilaaja tapahtuman tilaajalistalta. Tämä poikkeus tapaus on kuvattu sekvenssikaaviona kuviossa 10.



Kuvio 10. Tilaajalistan päivitys, kun tilaaja ei ole enää aktiivinen

6.2.2 Subscribe

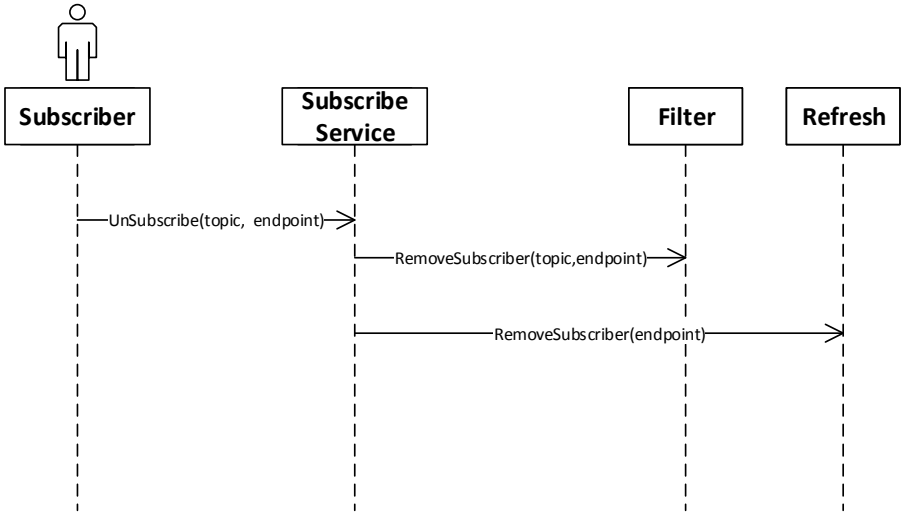
Tapahtumien tilaus tehdään Subscribe-operatiolla. Pyynnön saatuaan palvelu lisää tilaajan Filter- ja Refesh-luokan ylläpitämiin listoihin ja tarkistaa History-luokalta, onko tallennettuina mahdollisesti aiemmin vastaanotettuja tapahtumia ja välittää ne tilaajalle jos niitä löytyy. Kuviossa 11 käyttötapaus on kuvattu sekvenssikaaviona.



Kuvio 11. Tapahtuman tilaus palvelulta

6.2.3 Unsubscribe

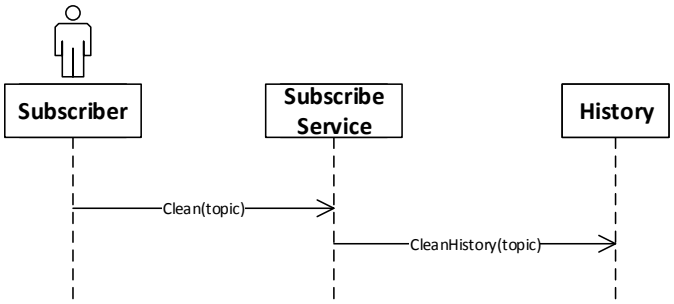
Aiheen tilaus perutaan UnSubscribe-operatiolla. Tämän vastaanotettuaan palvelu poistaa tilauksen sekä Filter- että Refresh-luokkien listoilta. Käyttötapaus on kuvattu sekvenssikaaviona kuviossa 12.



Kuvio 12. Tilauksen peruuttaminen

6.2.4 Clean

Asiakas voi halutessaan tyhjentää tietyn aiheen historiatiedot palvelimelta. Tämä tapahtuu Subscribe-palvelun Clear-operatiolla, jonka vastaanottaessaan palvelu tyhjentää halutun aiheen historian History-luokasta. Tämä toiminta on kuvattu kuviossa 13.

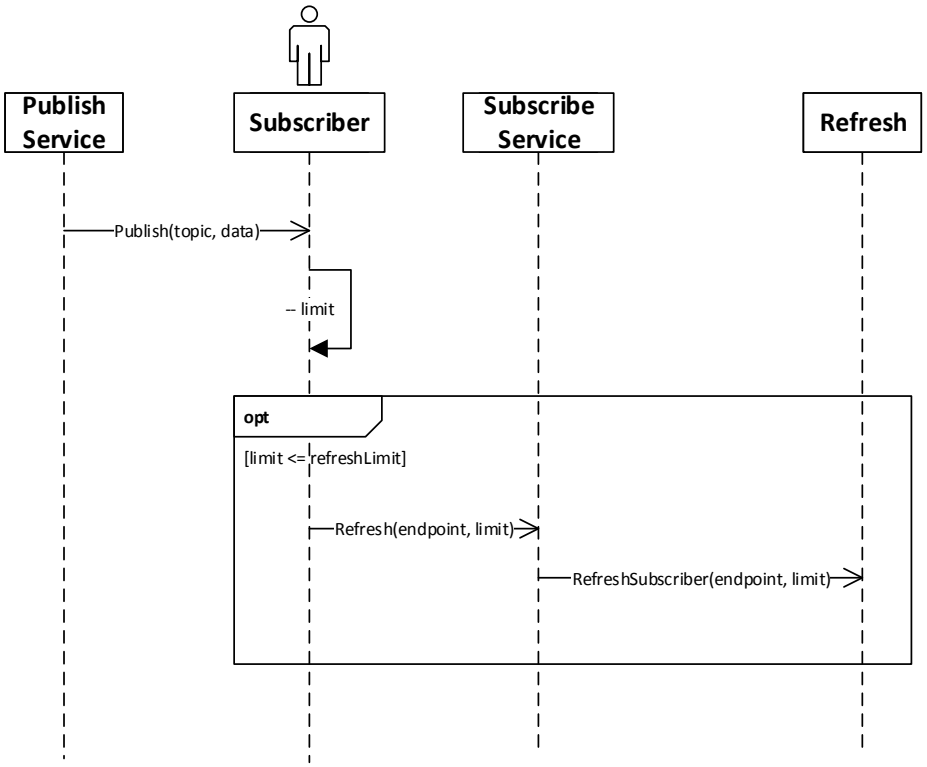


Kuvio 13. Historiatiedon tyhjentäminen

6.2.5 Refresh

Susbcriber-asiakas ylläpitää myös itse tietoa siitä, montako tapahtumaa se on tilaamistaan vastaanottanut. Kun tilatut tapahtumat on vastaanotettu, uudistetaan

tilaus Refresh-operatiolla. Pyynnön vastaanotettuaan Susbcribe-palvelu päivittää uuden määrän asiakaskohtaiseen listaansa Refresh-luokassa. Toiminta on kuvattu sekvenssikaaviona kuviossa 14.



Kuvio 14. Tilauksen virkistäminen

6.3 Sanomaprotokolla

Sanoman välitys asiakkaan ja palvelimen välillä on toteutettu käyttäen UDP-protokollaa, jossa sanoma lähetetään ASCII-merkkijonona. Itse sanomassa eri kentät on eroteltu toisistaan pilkulla. Subscribe-palvelu kuuntelee sanomia UDP-portissa 10001 ja Publish-palvelu kuuntelee sanomia portissa 10002. Asiakkaan lähettämät sanomat alkavat aina operaatiolla, jota seuraa tarvittava määrä parametreja. Publish-palvelun asiakkaalle lähettämät sanomat sisältävät ainoastaan dataa. Seuraavissa taulukoissa on eri väleillä käytetyt operaatiot ja niiden parametrit.

Taulukko 1. Sanomat Subscriber-asiakkaalta Subscribe-palvelulle.

Sanomat Subscriber-asiakkaalta Subscribe-palvelulle	Käyttötarkoitus
Subscribe,<topic>,<refreshlimit>	Tapahtumien tilaus palvelusta.
UnSubscribe,<topic>	Tapahtuma tilauksen peruutus.
Clear,<topic>	Tapahtuma historian tyhjennys.
Refresh,<topic>,<refreshlimit>	Tapahtuma tilauksen virkistys.

Taulukko 2. Sanoma Publisher-asiakkaalta Publish-palvelulle

Sanoma Publish-asiakkaalta Publish-palvelulle	Käyttötarkoitus
Publich,<date>,<topic>,<data>	Tapahtuman jakelu pyyntö palvelulle.

Taulukko 3. Sanoma Publish-palvelulta Subscriber-asiakkaille

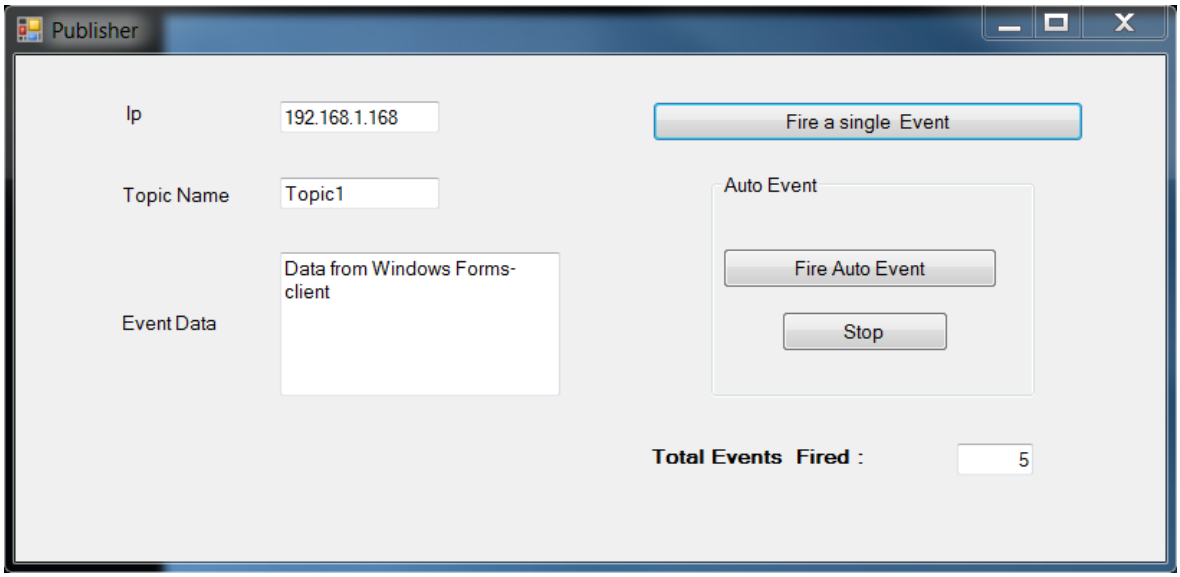
Sanoma Publish-palvelulta Subscriber-asiakkaille	Käyttötarkoitus
<date>,<topic>,<data>	Tapahtuman jakelu asiakkaille.

6.4 Käyttöliittymät

Subscriber- ja Publisher-asiakkaille toteutettiin tässä työssä useita erityyppisiä käyttöliittymiä. Graafiset käyttöliittymät toteutettiin Windows-Form-pohjaisena Windows-työasemia varten, sekä Javalla Android-puhelimia ja -tabletteja varten. Lisäksi toteutettiin merkkipohjainen komentorivikäyttöliittymä sekä C#- että Python-kielellä.

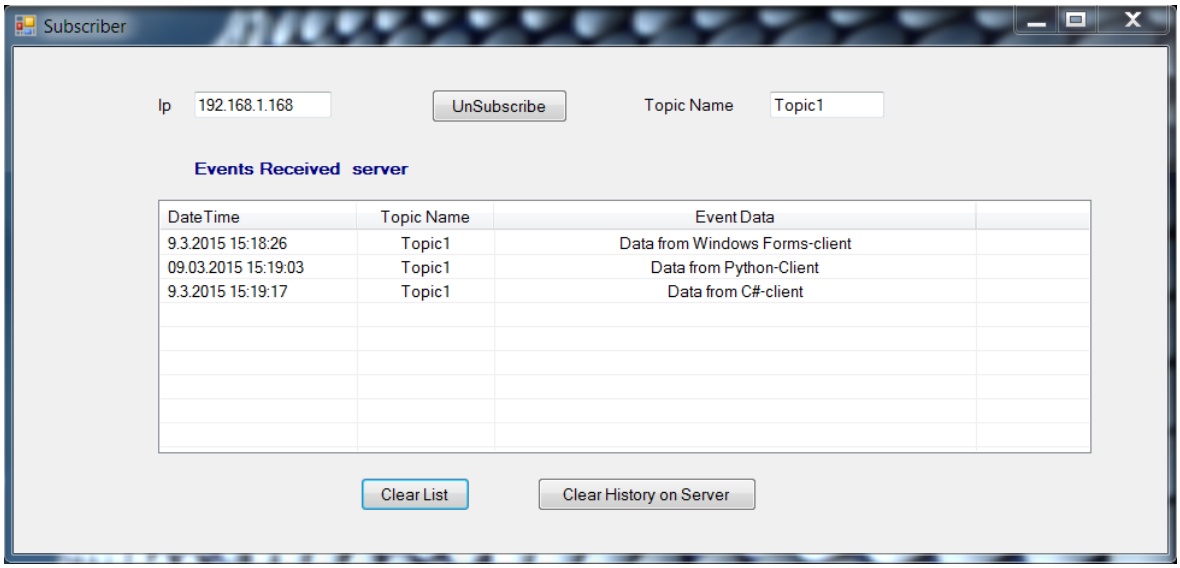
6.4.1 Windows Forms

Publisher-asiakasohjelmassa on parametrisoitu Publisher-palvelun IP-osoite, lähettävän tapahtuman aihe sekä varsinainen data. Viestin lähettäminen on mahdollista joko yksittäin tai sarjana jolloin tapahtumaa lähetetään sekunnin välein kunnes painetaan Stop-painiketta. Kuviossa 15 on kuvakaappaus Publisher-asiakasohjelman toteutuksesta.



Kuvio 15. Publisher-asiakas, Windows Forms -versio

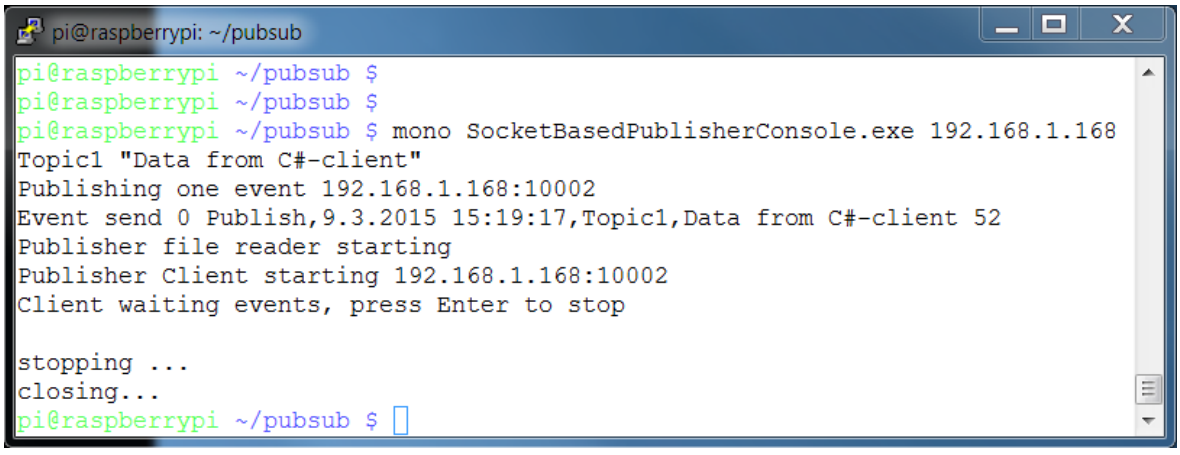
Subscriber-asiakasohjelmassa on parametrisoitu Publisher-palvelun IP-osoite ja vastaanotettavan tapahtuman aihe. Tapahtumien vastaanotto aloitetaan painamalla Subscribe-painiketta. Se vaihtuu painalluksen jälkeen UnSubscribe-painikkeeksi, jota painamalla tapahtumien vastaanotto lopetetaan. Vastaanotetut viestit kertyvät taulukkoon, joka on mahdollista tyhjentää paikallisesti ClearList-painikkeella. Clear History on Server -painike lähettää palvelulle pyynnön tyhjentää valitun aiheen mukainen tapahtumahistoria palvelimelta. Viestien vastaanotto ja tapahtumalistan päivitys toteutettiin käyttäen omaa säiettä, jotta viestien odottelu ei estä käyttöliittymän toimintaa. Kuviossa 16 on kuvakaappaus Subscriber-asiakasohjelman toteutuksesta.



Kuvio 16. Subscriber-asiakas, Windows Forms -versio

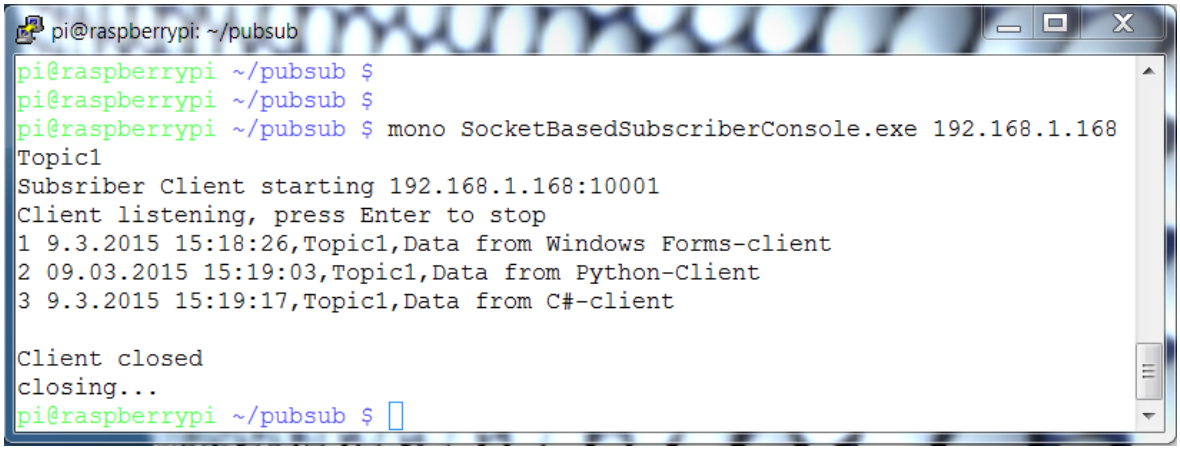
6.4.2 Komentorivi

C#-komentoriviasiakas ajetaan käyttämällä Mono-ohjelmaa. Asiakasohjelmalle annetaan kolme parametria: ensimmäinen on Publisher-palvelun IP-osoite, toinen on tapahtuman aihe, ja kolmas parametri on tapahtuman vapaavalintainen data. Lisäksi tähän konsoliasiakkaaseen toteutettiin tapahtumien lähettäminen tiedoston perusteella. Tiedostosta lähettäessään asiakasohjelma etsii tapahtumatiedostoa levyltä ja jos tämä tiedosto löytyy, lähetetään sen sisältö tapahtumina rivi kerrallaan Publish-palvelulle. Kuviossa 17 on kuvakaappaus Publisher-komentoriviasiakkaan testiajosta.



Kuvio 17. Publisher-asiakas, C#-komentoriviversio

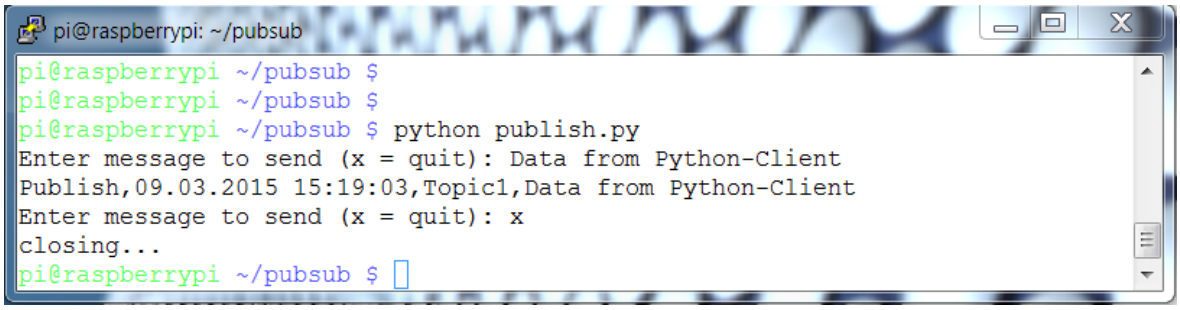
C#-komentoriviasiakas ajetaan myös käyttämällä Mono-ohjelmaa. Asiakasohjelmalle annetaan kaksi parametria: ensimmäinen on Pulisher-palvelun IP-osoite ja toinen on tapahtuman aihe. Tämän jälkeen asiakasohjelma tilaa palvelimelta parametrin mukaiset tapahtumat ja tulostaa näytölle vastaanottamansa tapahtumat niiden saapuessa. Asiakasohjelma vastaanottaa tapahtumia kunnes painetaan Enter-näppäintä. Näppäimen painalluksesta asiakas poistaa tapahtumatilauksensa palvelimelta. Kuviossa 18 on kuvaruutukaappaus Subscriber-komentoriviasiakkaan testiajosta.



Kuvio 18. Subscriber-asiakas, C#-komentoriviversio

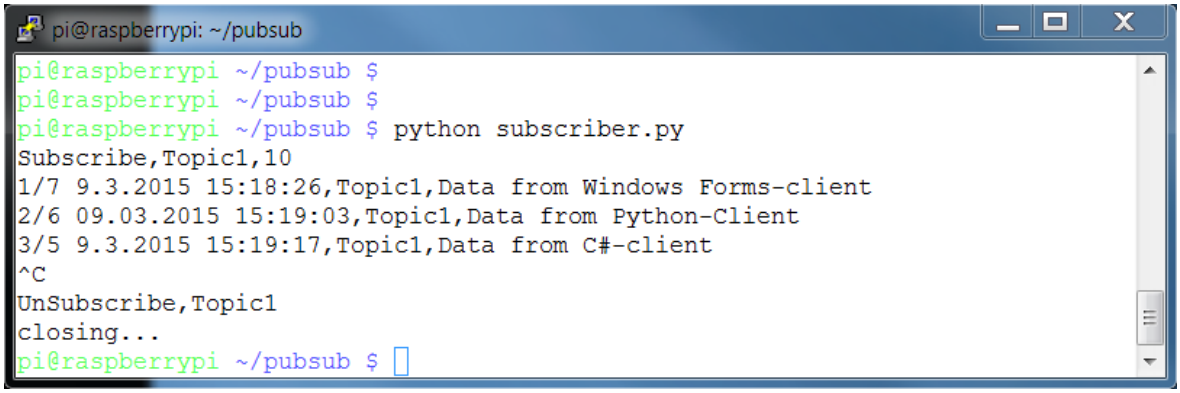
6.4.3 Python

Publisher-asiakkaan Python-versiossa Publish-palvelun IP-osoite ja tapahtuman aihe on kovakoodattu itse skriptiin. Nämä tiedot on muutettavissa tarvittaessa editorilla. Asiakasohjelma odottaa käynnistyessään käyttäjältä lähetettävää dataa ja sen saatuaan lähettää uuden tapahtuman palvelimelle, sekä tulostaa lähetetyn viestin. Asiakas suljetaan antamalla lähetettäväksi viestiksi “x”. Kuviossa 19 on kuvaruutukaappaus Publisher-komentoriviasiakkaan testiajosta.



Kuvio 19. Publisher-asiakas, Python-versio

Myös Subscriber-asiakkaan Python-versiossa Subscriber-palvelimen IP-osoite ja tapahtuman aihe on kovakoodattu itse skriptiin. Nämä tiedot on muutettavissa tarvittaessa editorilla. Asiakasohjelma tilaa käynnistyessään tapahtumat palvelimelta ja alkaa tulostaa niitä konsolille sitä mukaa kuin palvelin niitä lähettää. Kun asiakas-ohjelma pysäytetään Control-C-näppäimellä, se poistaa tilauksensa palvelimelta. Kuviossa 20 on kuvaruutukaappaus Publisher-komentoriviasiakkaan testiajosta.

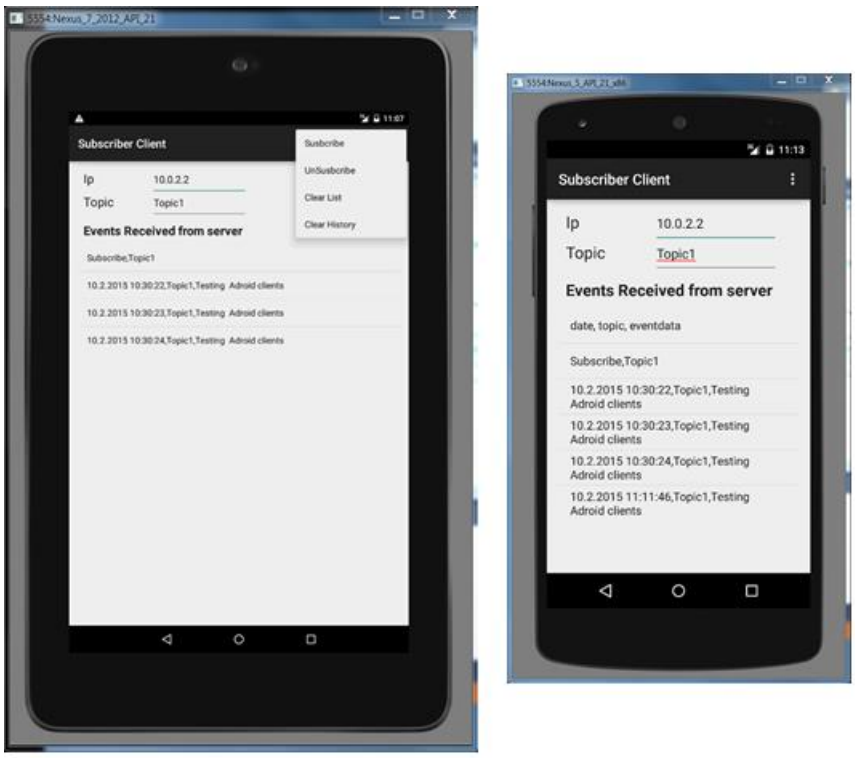


Kuvio 20. Subcriber-asiakas, Python-versio

6.4.4 Android-puhelin ja -tabletti

Subscriber-asiakasohjelmassa on parametrisoitu Publisher-palvelun IP-osoite ja vastaanotettavan tapahtuman aihe. Tapahtumien vastaanotto aloitetaan valitsemalla valikosta Subscribe-toiminto. Tapahtumien vastaanotto lopetetaan valitsemalla UnSubscribe-toiminto. Vastaanotetut viestit kertyvät taulukkoon, joka on mahdollista tyhjentää paikallisesti ClearList-toiminnolla. Clear History -toiminto lähettää palvelulle pyynnön tyhjentää valitun aiheen mukainen tapahtumahistoria

palvelimelta. Kuviossa 21 on kuvakaappaus Android-asiakkaan testauksesta käyttäen emulaattoria. Kuviossa IP-osoitteena oleva 10.0.2.2 on emulaattorin tuntema erikoisosoite, johon lähettävät viestit lähetetään ajoympäristön localhost-osoitteeseen. Viestien vastaanotto ja tapahtumalistan päivitys toteutettiin käyttäen omaa säiettä, jotta viestien odottelu ei estä käyttöliittymän toimintaa.



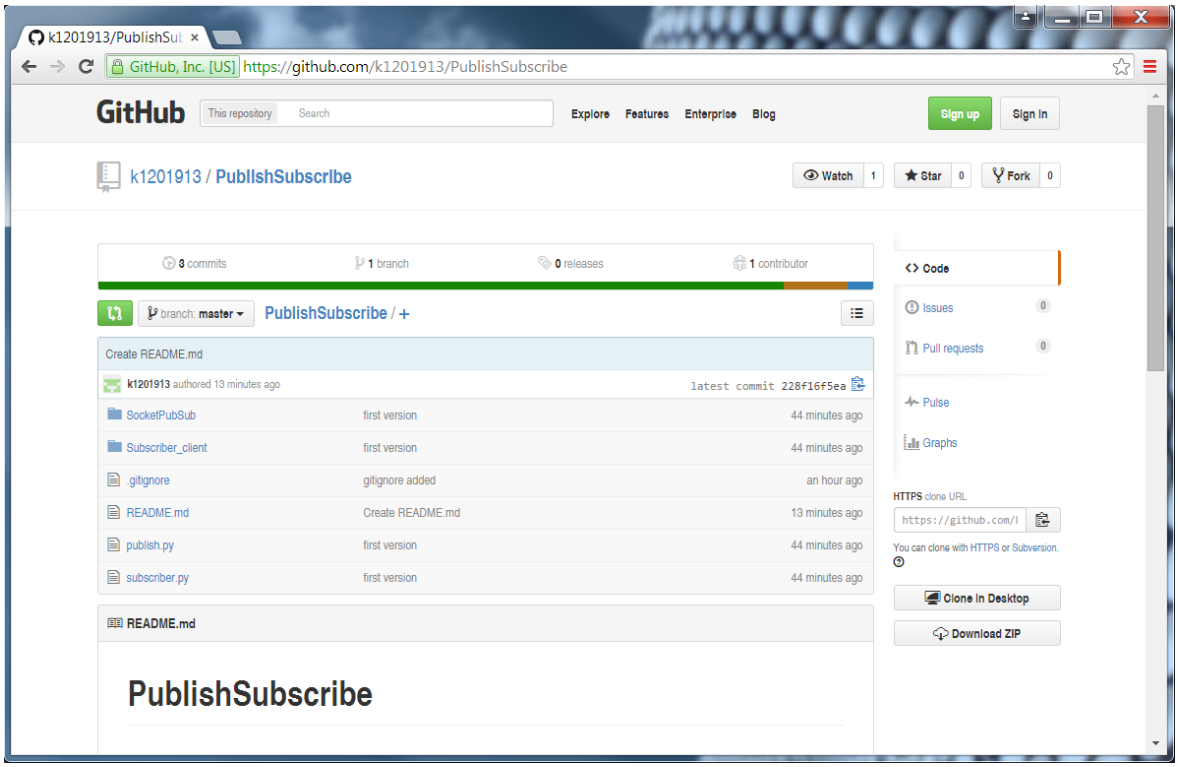
Kuvio 21. Subscriber-asiakas Android-tabletti ja -puhelinversio

6.5 Lähdekoodit GitHubissa

Prototyyppi toteutuksen lähdekoodit ovat julkisesti saatavilla GitHubissa. Koodit voi kloonata täällä omaan kuvauskantaansa git-komennolla.

git clone https://github.com/k1201913/PublishSubscribe.git

Vaihtoehtoisesti niitä voi selailla myös suoraan internet-selaimella samassa osoitteessa. Kuviossa 22 on kuvakaappaus internet-selaimen näkymästä GitHub-projektiin.



Kuvio 22. Lähdekoodit GitHub-projektissa.

SocketPubSub-hakemisto sisältää Visual Studio -ohjelmalla kirjoitetut C#-kieliset projektit. Subscriber_client-hakemisto sisältää Android Studio -ohjelmalla kirjoitetun Subscriber-asiakkaan. Lisäksi päätasolla löytyy Python-kieliset Subsciber- ja Publisher-asiakkaat.

7 TYÖN TULOKSET

7.1 Tulokset

Projektissa saatiin aikaan prototyyppitoteutus, jolla voidaan kokeilla tiedon siirtoa tietoja keräävältä Raspberry Pi -koneelta palvelimen kautta eri järjestelmissä toimiville asiakasohjelmille. Myös Mono-ohjelmisto saatiin toimimaan halutulla tavalla, ja siten tuli todennettua mahdollisuus ajaa Windows-ympäristössä käännettyä C#-koodia Linux-järjestelmässä.

Asiakasohjelmat saatiin onnistuneesti toimimaan Linuxissa, Windows-työasemissa sekä Androidissa. Alun perin mukaan kaavailtu Windows-phone ja -tablet Windows Store apps -sovellus pudotettiin pois työn sisällöstä Windows 8 -kehitysympäristön puutteen vuoksi.

Asiakasohjelmat testattiin sekä Python- että C#-kielisinä. Jos tiedonkeruu Rasberry PI-koneessa on toteutettavissa Python-kielisenä, niin myös tiedonvälitys eteenpäin on syytä toteuttaa samalla kielellä. Mono-ohjelmistolla ajettava C#-kielinen ohjelma tarjoaa tarvittaessa vaihtoehtoisen tavan tiedon esikäsittelyyn ja sen palvelimelle lähettämiseen.

Varsinaista todellista käyttöä varten käyttöliittymien käyttämät tekstit pitäisi päivittää vastaamaan käyttötarkoituksen termistöä. Myös tapahtuman kenttien määrä ja sisältö voi muuttua käyttötarkoituksen mukaan. Lisäksi nykyinen palvelinohjelman prototyyppi, joka käynnistyy konsoli tai Window Forms -ohjelmana, tulisi päivittää palvelimelle käynnistyväksi palveluksi.

7.2 Yhteenveto

Yhteenvetona voidaan todeta että Raspberry Pi -kone osoitti jälleen monipuolisuutensa. Kone on hintaansa nähden muuntautumiskykyinen alusta, jonka päälle voidaan rakentaa monenlaisia tiedonkeruusovelluksia. Raspberry Pi -kone tarjoaakin yhden edullisen tavan kerätä tietoa erilaisilta antureilta tai järjestelmistä ja siirtää tieto normaalissa internetverkossa jatkokäyttöön.

Työ oli mielenkiintoinen, koska siinä pääsi yhdistelemään monissa eri oppiaineissa kertyneitä taitoja. Työtä tehdessä tuli kerrattua Linuxin perusteet, C#-ohjelmointi, Python-ohjelmointi, Java-ohjelmointi sekä tutustuttua mielenkiintoiseen Mono-ohjelmistoon. Lisäksi työssä tutustuttiin uusimpaan versioon Android-kehitysympäristöstä, sen tarjoamiin työkaluihin ja ohjelman kehitykseen niitä käyttäen.

LÄHTEET

Android. Ei päiväystä. Android Studio. [www-dokumentti]. Android. [viitattu 10.2.2015]. Saatavana: http://developer.android.com/sdk/index.html

BinaryTides. 2012. Programming udp sockets in python. [www-dokumentti]. BinaryTides. [viitattu 19.11.2014]. Saatavana: http://www.binarytides.com/programming-udp-sockets-in-python/

Code Project. 2009. Topic-based Publish/Subscribe design pattern implementation in C# - Part I (Using socket programming). [www-dokumentti]. Code-Project. [viitattu 24.10.2014]. Saatavana: http://www.codeproject.com/Articles/34316/Topic-based-Publish-Subscribe-design-pattern-imple

Comer, D. E. 2002. TCP /IP. Helsinki: Edita Publishing Oy.

Dan's Website. Ei päiväystä. Raspberry Pi and Mono – Hello World! [www-dokumentti]. Dan's Website. [viitattu 24.10.2014]. Saatavana: http://logicalgenetics.com/raspberry-pi-and-mono-hello-world/

Ecma 2006. Standard ECMA-334 C# Language Specification. [www-dokumentti]. Ecma International. [viitattu 7.11.2014]. Saatavana: http://www.ecma-international.org/publications/standards/Ecma-334.htm

Ecma 2012. Standard ECMA-335 Common Language Infrastructure (CLI). [www-dokumentti]. Ecma International. [viitattu 7.11.2014]. Saatavana: http://www.ecma-international.org/publications/standards/Ecma-335.htm

Microsoft. Ei päiväystä. Design Patterns: List-Based Publish-Subscribe. [www-dokumentti]. Microsoft. [viitattu 24.10.2014]. Saatavana: http://msdn.microsoft.com/en-us/library/ms752254.aspx

Microsoft. Ei päiväystä. Socket Class. [www-dokumentti]. Microsoft. [viitattu 19.11.2014]. Saatavana: http://msdn.microsoft.com/en-us/library/system.net.sockets.socket(v=vs.110).aspx

Mono. Ei päiväystä. About Mono. [www-dokumentti]. Mono. [viitattu 7.11.2014]. Saatavana: http://www.mono-project.com/docs/about-mono/

Oracle. 2104. Writing a Datagram Client and Server. [www-dokumentti]. Oracle. [viitattu 10.2.2015]. Saatavana: http://docs.oracle.com/javase/tutorial/networking/datagrams/clientServer.html

Raspberry Pi Foundation. Ei päiväystä. Raspberry Pi. [www-dokumentti]. Raspberry Pi Foundation. [viitattu 23.10.2014]. Saatavana: http://www.raspberrypi.org/

Raspberry Pi Foundation. Ei päiväystä. What is a Raspberry Pi? [www-dokumentti]. Raspberry Pi Foundation. [viitattu 19.11.2014]. Saatavana: http://www.raspberrypi.org/help/what-is-a-raspberry-pi/

Raspberry Pi Foundation. Ei päiväystä. INSTALLING OPERATING SYSTEM IMAGES. [www-dokumentti]. Raspberry Pi Foundation. [viitattu 6.2.2015]. Saatavana: http://www.raspberrypi.org/documentation/installation/installing-images/README.md

Raspberry Pi Foundation. Ei päiväystä. Downloads. [www-dokumentti]. Raspberry Pi Foundation. [viitattu 7.2.2015]. Saatavana: http://www.raspberrypi.org/downloads/

Raspberry Pi Foundation. Ei päiväystä. INSTALLING OPERATING SYSTEM IMAGES USING WINDOWS. [www-dokumentti]. Raspberry Pi Foundation. [viitattu 8.2.2015]. Saatavana: http://www.raspberrypi.org/documentation/installation/installing-images/windows.md